Java-Einführungskurs

Informatik II für D-ITET
FS 2014, ETH Zürich

Simon Mayer
simon.mayer@inf.ethz.ch

Ziele / Überblick

- Vorbereitung auf die Übungen zu Informatik II
  + Vorstellung des Teams… ;-)

- Theorie
  - Java-Technologie und Sprache
  - Unterschiede: C++ vs. Java

- Praxis: Übungsblatt 0
  - HelloWorld.java
  - Erste Schritte mit Eclipse
  - JUnit4 für automatisiertes Testen

Simon Mayer
http://people.inf.ethz.ch/mayersi

institute for pervasive computing

RESEARCH GROUP FOR
Distributed Systems

Distributed Systems Group

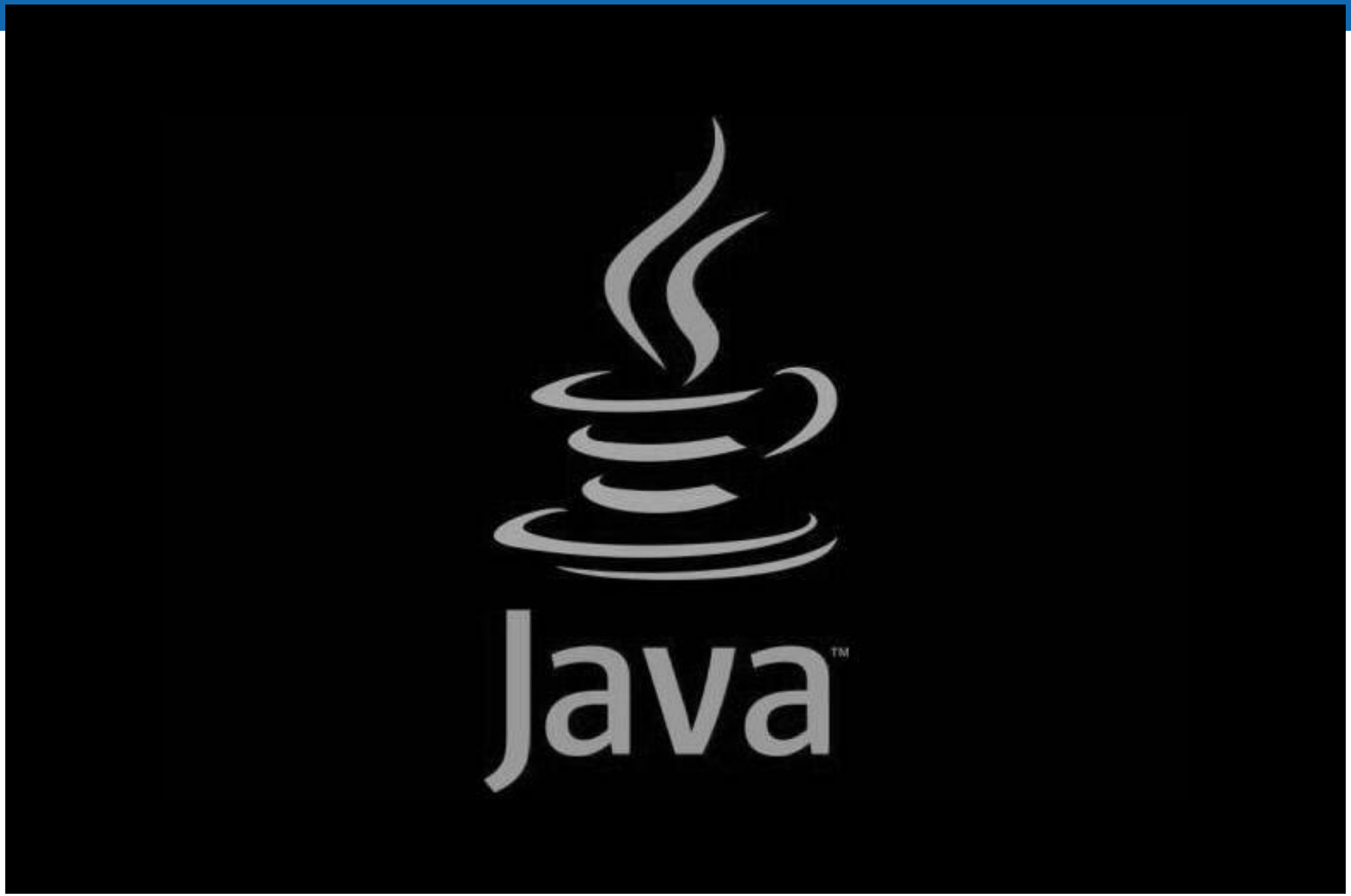
Institute for Pervasive Computing

Simon Mayer
http://people.inf.ethz.ch/mayersi

institute for pervasive computing

RESEARCH GROUP FOR Distributed Systems

Simon Mayer
http://people.inf.ethz.ch/mayersi



Simon Mayer
http://people.inf.ethz.ch/mayersi

Warum Java

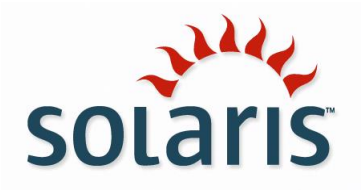
- Objektorientiert
- Einfacher als C++
- Umfangreiches Ökosystem: Tools, Bibliotheken, ...
- Virtuelle Maschine: „Compile once – Run everywhere“

Werdegang eines Java-Programms

Program.java **Quellcode**: Menschenverständlicher Text

Plattformunabhängigkeit: Java-Bytecode ist ohne Änderung auf jeder Architektur lauffähig, auf welcher eine Laufzeitumgebung installiert ist :-D



Simon Mayer
http://people.inf.ethz.ch/mayersi

Werdegang eines Java-Programms

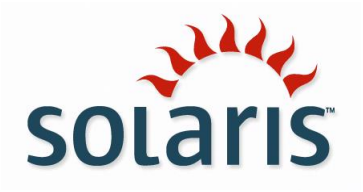
Program.java — **Quellcode**: Menschenverständlicher Text

↓ `javac Program.java` — Aufruf des **Java-Compilers**

Program.class — **Java-Bytecode**: Maschinenverständlicher Code

Plattformunabhängigkeit: Java-Bytecode ist ohne Änderung auf jeder Architektur lauffähig, auf welcher eine Laufzeitumgebung installiert ist :-D



Simon Mayer
http://people.inf.ethz.ch/mayersi

Werdegang eines Java-Programms

Program.java — **Quellcode**: Menschenverständlicher Text

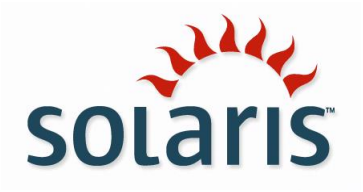
↓ `javac Program.java` — Aufruf des **Java-Compilers**

Program.class — **Java-Bytecode**: Maschinenverständlicher Code

↓ `java Program` — Ausführen mittels **virtueller Maschine**

Plattformunabhängigkeit: Java-Bytecode ist ohne Änderung auf jeder Architektur lauffähig, auf welcher eine Laufzeitumgebung installiert ist :-D



Simon Mayer
http://people.inf.ethz.ch/mayersi

Werdegang eines Java-Programms

Program.java — **Quellcode**: Menschenverständlicher Text

↓ `javac Program.java` — Aufruf des **Java-Compilers**

Program.class — **Java-Bytecode**: Maschinenverständlicher Code

↓ `java Program` — Ausführen mittels **virtueller Maschine**

Plattformunabhängigkeit: Java-Bytecode ist ohne Änderung auf jeder Architektur lauffähig, auf welcher eine Laufzeitumgebung installiert ist :-D




Simon Mayer
http://people.inf.ethz.ch/mayersi

Die Java-Technologie

- Java-Laufzeitumgebung: JRE
  - Hauptbestandteil ist das Programm `java`
  - Eine Art „Virtueller Computer“
  - Java Virtual Machine (JVM)
  - Standardklassen und weitere Programmbibliotheken
  - JRE-Editionen: Java SE, Java ME, Java EE

- Java-Entwicklungswerkzeug: JDK
  - Enthält die Programme `java`, `javac`, `javap...`

Java-Ökosystem

- Standardbibliothek
  - Datenstrukturen (List, Map,...), Algorithmen (Sort,...), Kryptographie, Kommunikation, graphische Benutzerschnittstellen
- Zusätzliche Bibliotheken
  - Datenbanken, Web-Server,...
- Integrierte Entwicklungsumgebung (IDE)
  - Editor + Compiler + Debugger + Projektverwaltung + …
  - Beispiel: **Eclipse**, NetBeans, IntelliJIDEA, …

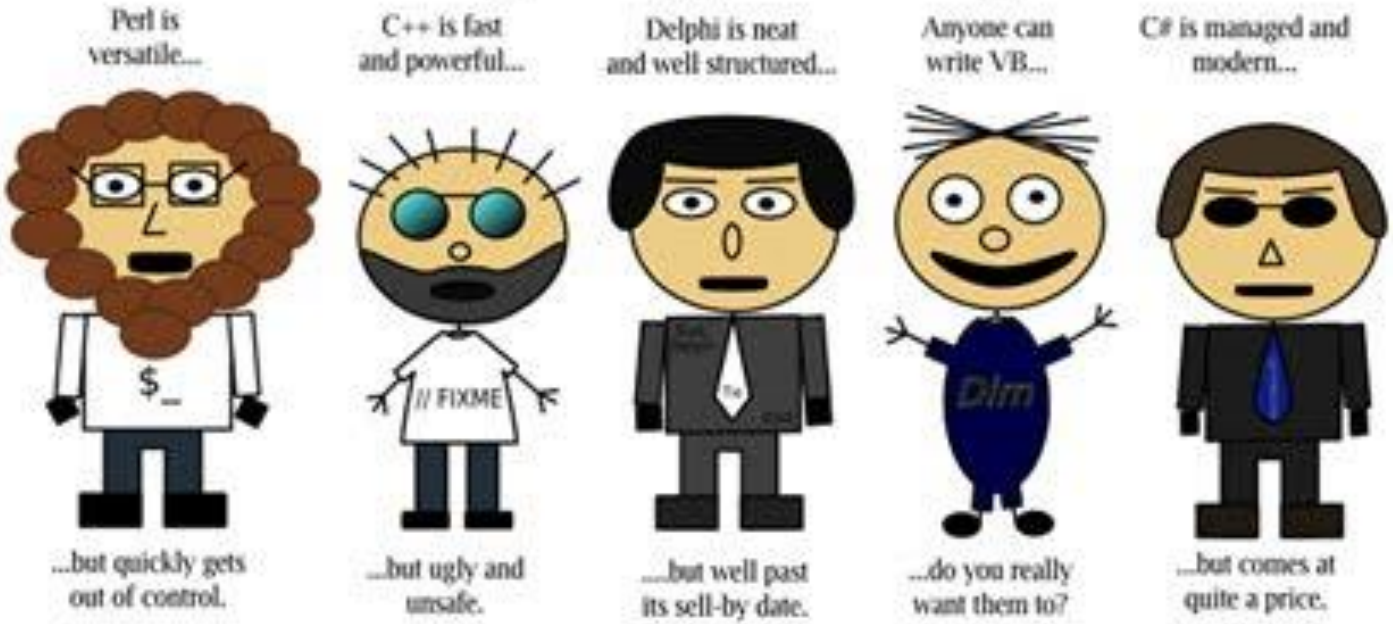
Java-Ökosystem

- Javadoc
  - Dokumentation durch *strukturierte* Kommentare
- Unit Testing (JUnit)
  - Bestandteil aller Übungen
  - Automatisiertes Testen des Codes
  - Generierung von Testberichten

Java-Sprache: Versionen

- JDK 1.0 (1996)
- JDK 1.1 (1997, z.B. Paketierung als .jar-Dateien)
- J2SE 1.2 (1998, z.B. Just-In-Time Compiler, Grafik)
- J2SE 1.3 (2000)
- J2SE 1.4 (2002, z.B. + Assertions und Server)
- Java 5.0 (2004, z.B. + Generics, Annotationen)
- Java 6.0 (2006)
- **Java 7.0** (2011, z.B. neue Filesystem-API, IPv6)
- Java 8.0 (2013)

Which language will you use?



Pick the right one for the job:java!

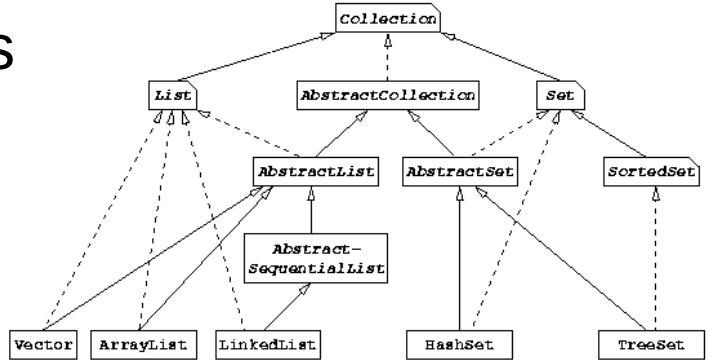
HelloMachine.java

```
package ch.ethz.itet.info2;

public class HelloMachine {

 public static void main(String [] args) {
    String userName = args[0];

    HelloMachine myHelloMachine = new HelloMachine();

    myHelloMachine.sayHello(userName);
 }

 public void sayHello(String name) {
    System.out.println("Hello, " + name + "!");
 }
}
```

Beachten: Funktionssignaturen, besonders keyword „static“; Parameter; System.out.println(...); Objekte;

Übersicht

- Pakete und Organisation von Java-Code
- Zugriffsrechte
- Typen, Vererbung, Polymorphismus
- Fehlerbehandlung, Stack Traces



Java-Sprache: Pakete

- Klassen können Teil eines „Pakets“ sein
  - Definition: `package myPackage;`
  - Navigation per Punkt: `.`
- Nutzen
  - Vermeidung von Namenskollisionen
  - Kompakterer, einfacher lesbarer Code
- Pakethierarchie wird auf Verzeichnisbaum abgebildet!

`ch.ethz.itet.info2` entspricht `ch/ethz/itet/info2`

Simon Mayer
http://people.inf.ethz.ch/mayersi

Java-Sprache: Organisation

- Jede öffentliche Klasse steht in ihrer eigenen *gleichnamigen .java*-Datei
  - Öffentliche Klasse: public class HelloMachine { … }
  - Nicht-öffentliche Klasse: private class HelloHelper { … }

- Pro (öffentlicher oder nicht-öffentlicher) Klasse generiert *javac* eine *.class*-Datei

Java-Sprache: Bibliotheken

- Sammeln von Paketen in .jar-Dateien (java archives)
- Zugriff auf Bibliothek
  - Bekanntmachen des Namens: `import` ...
  - Namen aus dem eigenen Paket sind immer bekannt
- Standardbibliothek steht automatisch zur Verfügung und muss nicht importiert werden
  - Dokumentation: http://docs.oracle.com/javase/7/docs/api/

Demo: Organisation, Pakete, Zugriff

- `Public.java`
  - Klasse „Public“ in Package „demo1“
  - `public static void main() { … }`
  - Benutzt Klasse ExtendedPublic

- `ExtendedPublic.java`
  - Öffentliche Klasse „ExtendedPublic“
  - + Private Klasse „Private“
  - In Paket „demo2“
  - `public void foo() { … }`

Java-Sprache: Zugriffsrechte

- `private`
- `public`
- `protected`
- `package`

- Keine `friends`

Java-Sprache: Primitive Typen

- Primitive Typen können auf dem Stack angelegt werden, ihre Instanzen sind **keine Objekte**!

```
boolean
byte, short, int, long
float, double
char
```

- Korrespondierende Klassen, z.B. `int vs. Integer`
  - Werden, wie alle Objekte, als Referenzen by Value übergeben (mehr dazu in Übung 3) und am Heap allokiert

Java-Sprache: Überall Objekte!

- Objekt: Instanz einer Klasse
- Zugriff ausschliesslich über Referenzen!
- Erzeugung mit `new`

```
ExtendedPublic eP = new ExtendedPublic();
```

- Entfernung durch Garbage Collector, **kein `delete`**

Java-Sprache: Vererbung

- Java bietet keine Mehrfachvererbung!
  - Stattdessen: Schnittstellen (`interface`)
  - Eine Klasse kann mehrere Interfaces implementieren
  - Mehr dazu in Übung 6

- Polymorphismus: Funktionen sind grundsätzlich virtuell!
  - Wenn: `Car extends Vehicle`
  - Dann: `Car.getSpeed()` überdeckt `Vehicle.getSpeed()`

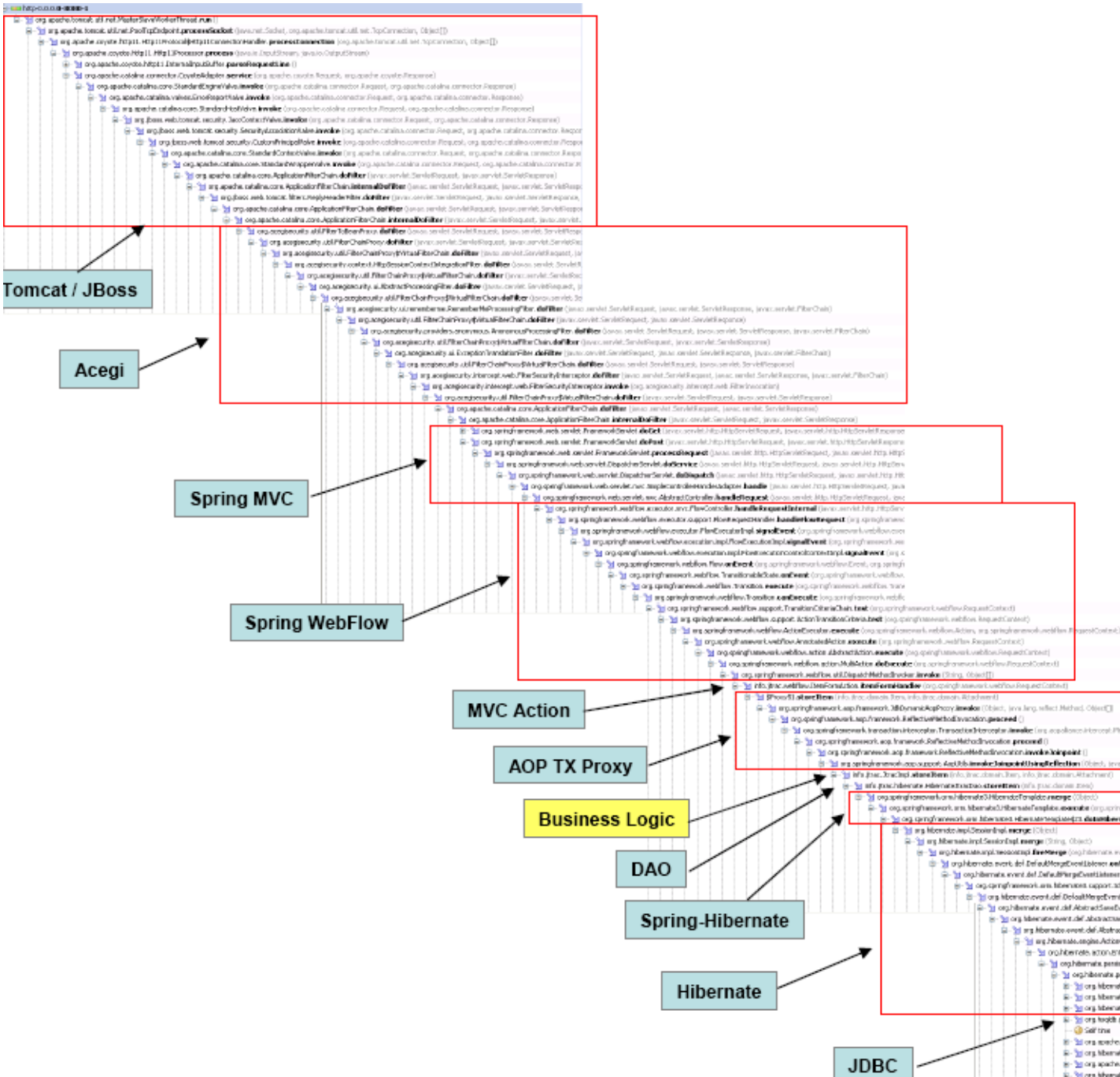
- Funktionen und Klassen können `abstract` sein
  - Abstrakte Klassen können nicht instanziiert werden

Java-Sprache: Fehler und Stack Traces

- Stack Traces ermöglichen das Zurückverfolgen von Fehlern zu ihrem Ursprung (+ Zeilennummern)

- Siehe Demo...

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
at StackTraceDemo.method2(StackTraceDemo.java:20)
at StackTraceDemo.method1(StackTraceDemo.java:12)
at StackTraceDemo.main(StackTraceDemo.java:7)
```


Tomcat / JBoss
Acegi
Spring MVC
Spring WebFlow
MVC Action
AOP TX Proxy
Business Logic
DAO
Spring-Hibernate
Hibernate
JDBC

Java-Sprache: Fehlerbehandlung

- **Error:** "indicates serious problems that a reasonable application should not try to catch."
  - Beispiel: `OutOfMemoryError`

- **Exception:** "indicates conditions that a reasonable application might want to catch."
  - Beispiel: `FileNotFoundException`

- Abfangen mit `try/catch`, werfen mit `throw`
  - Nicht abgefangene Exceptions (und Errors) führen zum Programmabbruch!
  - Mehr dazu in Übung 1 und 2


jokeOverflow.com
EXERCISE
Some motivation required.

Übungsblatt 0: Aufgabe 1

- HelloWorld mit Texteditor

- HelloWorld in Eclipse
  - Falls schon installiert: Super!
  - Sonst: Links zu Dokumentation auf Vorlesungswebsite

Übungsblatt 0: Aufgabe 1 und Eclipse

- Import der Übungsdaten in Eclipse
  - File – New – Java Project – [Projektnamen Eingeben] – Next – Link Additional Source – Übungsordner auswählen (z.B. „u0“) - Finish
- Neue Bibliothek einbinden
  - Rechtsklick auf Projekt – Build Path – Configure Build Path – „Libraries“
  - Einbinden von JARs: „Add (external) JARs“
  - Einbinden von JUnit4: „Add Library“
- Ausführen von HelloWorld

Übungsblatt 0: Aufgabe 2

- Erstes Java-Programm: Signum-Funktion

Simon Mayer
http://people.inf.ethz.ch/mayersi

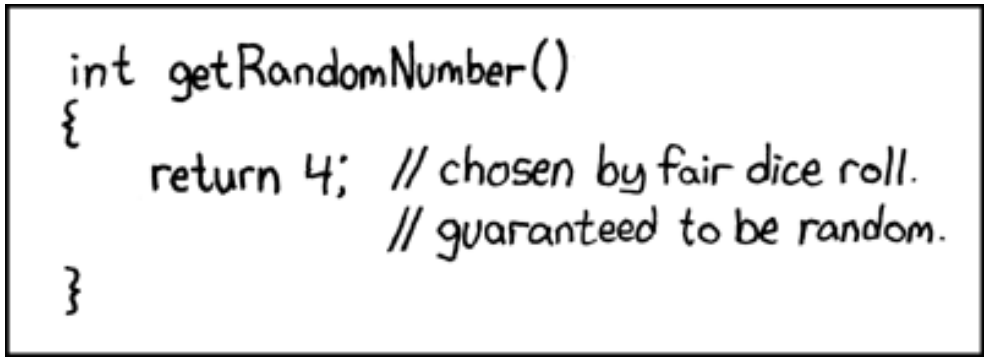
Übungsblatt 0: Aufgabe 3

- Automatisiertes Testen mit JUnit4
- … aus der Kommandozeile (wird nicht gezeigt…)
- … in Eclipse

Übungsblatt 0: Aufgabe 4

- Modellbildung, für zu Hause...

Simon Mayer
http://people.inf.ethz.ch/mayersi


int getRandomNumber()
{
return 4; // chosen by fair dice roll.
// guaranteed to be random.
}

Fragen?